fantasy-baseball-pipeline

Chris Toomey

Introduction

Each year, millions compete in an effort to best strangers and friends alike in the game of fantasy baseball, whether for bragging rights, money, or some other prize. Fielding a team, however, can be extremely difficult due to the huge amount of uncertainty and wide variety of data to sort. Some experts even make a living by writing columns about the *Top 100 Fantasy Baseball Players* or the *Top 10 Fantasy First Basemen*, and many novices take these assessments to heart when drafting, assuming that the expert has provided reliable research for them.

However, these static lists present a potential problem: is it always wise to take the next best player by rank alone, rather than a player who may fill a need? Similarly, if a certain position is being taken off the board quickly, should you scoop up a lesser player in that position, or ignore the league trend and stay your course?

In order to test these hypotheses, I

- constructed a dynamic model to update after each player is selected in order to account for my team's positional needs as well as league wide drafting trends,
- compared my model to two different models: one that selected the next best player based off of ESPN's Top 300 Fantasy Players list, and another that used the same data I did, but did not update dynamically.

ESPN Rotisserie (ROTO)

To test my hypotheses, I used ESPN's Rotisserie format. ROTO is one of the most common scoring formats for fantasy baseball. At the end of the season, points are totaled between categories specified before the season. For my pipeline, I kept traditional statistical categories:

- for batters, these include home runs (HR), average (AVG), on base plus slugging (OPS), stolen bases (SB), strikeouts (K), walks (BB), and runs batted in (RBI)
- for pitchers, these include strikeouts (K), wins (W), saves (S), walks (BB), earned runs (ERA), and walks/hits per inning pitched (WHIP)

Although I tried to keep categories that were most common among fantasy baseball leagues, the model can be used with any conceivable categories given sufficient projections.

The rosters include: one catcher (C), one first baseman (1B), one second baseman (2B), one shortstop (SS), one third baseman (3B), one middle infielder filled by a 2B/SS, one corner infielder filled by a 1B/3B, five outfielders (OF), one utility (UTIL) spot filled by any position, and three bench (BE) spots -- those who are on your bench are players who are on your roster but not in your starting lineup.

Data

The data I am using for my model is from the Steamers projections for the 2020 season. The other algorithms select the next best player on either ESPN's Top 300 Fantasy Players List by Tristan Cockcroft (an ESPN fantasy baseball writer) or using Steamers projections, depending on which I want to compare to my model. Since pitchers would essentially only add an additional

position, I only included position players and their categories. A similar test could be used with any positional and categorical restraints. Additionally, any constraints on categories, the number of slots per position, number of teams, or data included could have been used for these tests.

The maximum amount of games the model could use and still have enough players for drafting was 98. This was limited mostly by the catching position.

Model

The first step of the model is to take all players and normalize them against their respective positions using their categorical statistics. For example, all first basemen were compared to eligible (projected to start the minimum number of games specified) first basemen in all categories and given a z-score.

These z-scores were then averaged for each player over all categories for an overall average score. Then, all players were put in a single database, and given a new z-score based on their average score compared to all eligible players in the league. The model is greedy, so it chooses the best player from the database. Then, the model first redistributes the database with the selected player taken out, and, using an input discount factor, multiplies the scores of players of the same position as the drafted players by the discount factor. This allows players from positions that the drafter has not drafted from to sift upwards in the model, thus differentiating it from a model that only takes the best player each time regardless of other factors. If the discount factor is 1, the model will not shift the database.

Results

The categories for testing were

$$[R, H, RBI, BB, SO, SB],$$

the spots were

$$\{C:1, 1B:2, 2B:2, 3B:2, SS:2, OF:5, UTIL:1\}$$

There were 10 teams (1 user and 9 opponents). The algorithm was run 30 times to average the results, where the database each team chose from stayed the same, but the draft order was randomized each time. The number of minimum games were

$$[50, 60, 70, 80, 90, 95, 98],$$

where any more than 98 games as the minimum did not allow the user's database enough spots to finish the draft. A 95% confidence interval was created, and in the table, the mean is the average score per team (where the bolded score is the score using the model), lower is the lower bound of the confidence interval, and upper is the upper bound of the confidence interval. The results were as follows:

Discount	Number of Games	Mean/Lower/Upper	Scores
0.5	50	Mean	[**33.8**, 33.0, 31.2, 34.3, 31.7, 33.1, 33.1, 34.0, 32.1, 31.3]
0.5	50	Lower	[33.1, 30.7, 28.7, 31.8, 29.0, 30.5, 30.3, 31.3, 29.2, 28.2]
0.5	50	Upper	[34.5, 35.4, 33.7, 36.8, 34.4, 35.8, 35.8, 36.8, 35.0, 34.4]
0.5	98	Mean	[**38.7**, 31.1, 32.4, 30.2, 30.9, 31.2, 34.5, 34.0, 32.9, 32.8],
0.5	98	Lower	[37.7, 29.0, 30.4, 27.8, 28.6, 28.9, 32.0, 31.1, 30.7, 30.6]
0.5	98	Upper	[39.6, 33.2, 34.5, 32.5, 33.1, 33.6, 37.0, 36.9, 35.0, 35.0]
0.7	50	Mean	[**31.8**, 32.9, 33.0, 34.8, 33.3, 31.1, 33.2, 31.2, 34.1, 32.4]
0.7	50	Lower	[31.1, 30.7, 30.6, 32.9, 30.8, 28.4, 31.2, 28.7, 31.5, 30.2]
0.7	50	Upper	[32.5, 35.1, 35.3, 36.8, 35.8, 33.9, 35.2, 33.7, 36.7, 34.6]
0.7	98	Mean	[**42.9**, 31.8, 31.2, 31.8, 32.9, 29.5, 33.2, 33.3, 30.1, 32.0]
0.7	98	Lower	[41.5, 29.1, 28.8, 29.7, 30.5, 26.1, 30.8, 30.4, 28.3, 29.2]
0.7	98	Upper	[44.4, 34.5, 33.7, 33.8, 35.3, 32.8, 35.7, 36.2, 31.9, 34.8]
0.8	50	Mean	[**34.3**, 32.0, 32.2, 31.3, 32.4, 32.9, 32.5, 33.9, 33.3, 32.8]
0.8	50	Lower	[33.9, 29.8, 29.8, 29.0, 30.2, 30.6, 30.2, 32.0, 30.7, 30.4]
0.8	50	Upper	[34.8, 34.2, 34.6, 33.7, 34.5, 35.2, 34.8, 35.8, 35.9, 35.2]

Discount	Number of Games	Mean/Lower/Upper	Scores
0.8	98	Mean	[**42.1**, 30.3, 32.6, 32.9, 30.9, 33.1, 34.2, 31.7, 30.5, 30.1]
0.8	98	Lower	[41.0, 27.4, 30.0, 30.1, 28.6, 30.9, 31.3, 28.6, 27.9, 27.2]
0.8	98	Upper	[43.2, 33.1, 35.3, 35.6, 33.2, 35.4, 37.1, 34.7, 33.0, 33.0]
0.9	50	Mean	[**33.9**, 33.3, 31.9, 32.3, 33.3, 32.3, 32.5, 34.3, 32.5, 31.9]
0.9	50	Lower	[33.3, 30.6, 30.0, 30.0, 30.9, 29.6, 30.1, 31.6, 30.0, 29.4]
0.9	50	Upper	[34.5, 36.1, 33.8, 34.7, 35.7, 35.1, 35.0, 36.9, 35.0, 34.3]
0.9	98	Mean	[**39.8**, 30.8, 33.2, 30.0, 31.7, 34.1, 31.8, 33.6, 31.4, 32.0]
0.9	98	Lower	[38.9, 28.4, 31.0, 27.6, 28.9, 31.4, 29.4, 31.2, 28.7, 29.6]
0.9	98	Upper	[40.7, 33.2, 35.3, 32.4, 34.4, 36.8, 34.3, 36.1, 34.1, 34.5]
0.95	50	Mean	[**31.7**, 33.9, 32.0, 31.0, 32.9, 31.4, 34.1, 34.6, 31.8, 34.7]
0.95	50	Lower	[31.0, 31.4, 29.9, 29.1, 30.5, 29.3, 32.0, 32.3, 29.6, 32.6]
0.95	50	Upper	[32.4, 36.3, 34.2, 33.0, 35.2, 33.6, 36.1, 36.9, 34.0, 36.8]
0.95	98	Mean	[**40.7**, 31.7, 32.4, 31.8, 31.6, 32.0, 31.0, 34.1, 30.4, 32.6]
0.95	98	Lower	[39.7, 29.4, 30.0, 29.7, 29.7, 29.5, 29.0, 31.9, 28.3, 29.9]
0.95	98	Upper	[41.8, 34.1, 34.9, 34.0, 33.5, 34.5, 33.0, 36.4, 32.4, 35.3]
0.99	50	Mean	[**32.3**, 33.4, 33.6, 33.5, 33.6, 33.4, 32.3, 32.4, 31.7, 32.1]
0.99	50	Lower	[32.0, 31.6, 31.5, 31.5, 31.5, 31.4, 30.1, 30.3, 29.2, 29.9]
0.99	50	Upper	[32.5, 35.2, 35.7, 35.5, 35.7, 35.4, 34.5, 34.6, 34.1, 34.4]

Discount	Number of Games	Mean/Lower/Upper	Scores
0.99	98	Mean	[**38.5**, 32.0, 33.2, 33.2, 32.5, 31.9, 29.7, 34.9, 31.8, 31.3]
0.99	98	Lower	[37.9, 29.1, 30.8, 30.5, 30.1, 30.0, 27.1, 31.5, 29.5, 29.0]
0.99	98	Upper	[39.0, 34.9, 35.6, 35.9, 34.9, 33.7, 32.3, 38.3, 34.1, 33.6]
1.0	50	Mean	[**31.9**, 33.1, 33.9, 32.5, 33.8, 32.3, 33.9, 31.6, 31.1, 34.3]
1.0	50	Lower	[31.4, 30.7, 31.2, 30.4, 31.7, 30.0, 32.3, 29.6, 29.1, 32.2]
1.0	50	Upper	[32.5, 35.6, 36.5, 34.5, 35.9, 34.6, 35.6, 33.5, 33.0, 36.5]
1.0	98	Mean	[**38.5**, 30.3, 32.8, 32.1, 31.2, 32.3, 34.6, 33.7, 31.5, 32.1]
1.0	98	Lower	[38.0, 27.8, 30.2, 29.5, 28.4, 30.1, 32.3, 31.0, 28.8, 29.7]
1.0	98	Upper	[39.0, 32.8, 35.4, 34.7, 34.0, 34.5, 37.0, 36.4, 34.1, 34.5]
1.5	50	Mean	[**31.5**, 32.8, 31.4, 32.7, 32.6, 31.4, 33.0, 33.7, 34.7, 34.7]
1.5	50	Lower	[31.2, 30.7, 29.2, 30.9, 30.7, 29.6, 30.9, 31.8, 32.9, 32.7]
1.5	50	Upper	[31.8, 34.9, 33.7, 34.5, 34.4, 33.2, 35.0, 35.6, 36.4, 36.7]
1.5	98	Mean	[**39.3**, 32.8, 32.7, 31.5, 33.2, 31.8, 31.4, 33.1, 32.2, 29.5]
1.5	98	Lower	[38.1, 30.5, 30.2, 28.7, 30.7, 29.3, 28.1, 30.9, 29.7, 26.7]
1.5	98	Upper	[40.5, 35.2, 35.3, 34.3, 35.7, 34.2, 34.6, 35.3, 34.6, 32.3]

Clearly, there is a large upwards trend as a higher game minimum is added. Regardless of the discount factor, going from 50 to 98 games puts the user's total points from indistinguishable from other teams' total points to consistently higher. Additionally, though they were not included for space, the user's total points climbs slowly with each iterative increase in games minimum. So, for example, for any discount factor, having 70 games instead of 60 games made the model perform better. This is likely because with a higher game minimum, players are being normalized against better players, since those are the players projected to start more games. This is important for positions such as catcher, where the best catcher receives a huge boost when

eligible players only need 50 games, since few catchers play many games, but many play some games.

The model seems to perform best with discount factors of 0.7, 0.8, and 0.9, with all their means being above 40 with 98 games. In order to test if these perform better than picking the best player regardless of position, we can compare these with the discount factor of 1.0. These next tests take place with the specified discount factors, and this time with 100 trials and a confidence interval of 0.99. The results were as follows:

Discount	Number of Games	Mean/Lower/Upper	Scores
0.7	98	Mean	[**42.1**, 33.4, 32.3, 32.2, 32.2, 30.8, 31.4, 32.2, 30.6, 31.2]
0.7	98	Lower	[41.3, 31.5, 30.5, 30.4, 30.4, 29.0, 29.5, 30.3, 28.6, 29.3]
0.7	98	Upper	[43.0, 35.4, 34.1, 33.9, 34.1, 32.7, 33.3, 34.1, 32.5, 33.2]
0.8	98	Mean	[**42.1**, 32.6, 30.2, 31.9, 32.2, 32.3, 31.3, 31.2, 33.0, 32.0]
0.8	98	Lower	[41.3, 30.6, 28.2, 30.1, 30.3, 30.4, 29.5, 29.1, 31.2, 30.1]
0.8	98	Upper	[43.0, 34.6, 32.1, 33.7, 34.0, 34.1, 33.2, 33.2, 34.8, 33.9]
0.95	98	Mean	[**39.7**, 32.2, 32.6, 33.0, 31.6, 32.3, 31.4, 31.7, 32.2, 31.8]
0.95	98	Lower	[39.1, 30.4, 30.8, 31.4, 29.9, 30.7, 29.6, 30.0, 30.6, 29.9]
0.95	98	Upper	[40.3, 33.9, 34.4, 34.6, 33.3, 34.0, 33.3, 33.4, 33.9, 33.6]
1.0	98	Mean	[**38.7**, 32.2, 32.8, 32.0, 32.1, 32.9, 32.6, 32.2, 31.6, 31.9],
1.0	98	Lower	[38.2, 30.8, 30.9, 30.6, 30.5, 31.5, 30.7, 30.7, 30.1, 30.1]
1.0	98	Upper	[39.2, 33.5, 34.6, 33.4, 33.7, 34.3, 34.4, 33.8, 33.2, 33.7]

Discount factors of 0.7 and 0.8 are above the two higher factors, with means of 42.1 each, whereas discount factors of 0.95 and 1.0 produce scores of 39.7 and 38.7 respectively.

Limitations

The original goal of this project was to draft a large sample of teams using the model created versus ESPN's top listed players and at the end of a regular season, see how the drafted teams compared. Injuries may have produced problems, but the hope was that through a large enough sample, this may have been mitigated (though likely not completely, as the user and opponent

are likely to choose the same or almost the same players in each iteration). However, due to the coronavirus, an MLB season now seems unlikely, and even if it does occur, it is certainly not representative of a normal season (due to game limits, players potentially choosing to sit out, reduced travel, etc.). So, instead, the testing needed to be done on projections. While this makes for a good heuristic, it does not allow the model to be tested in the real world, and it is possible, for example, that ESPN has a drastically different projection system from other systems, but actually projects player totals better. Until an MLB season is played, however, this cannot be fully tested.

Conclusion

For a 10 person league, allowing a discount factor of 0.7 or 0.8 certainly seems to be better than picking the best player available, at least when compared to expert opinion at ESPN. They seem to perform equally well, with a mean score of 42.1 ± 0.1. A discount factor of 1, meaning selecting the best player available from a static database, produces a score of 38.7 ± 0.5. Both are well above opponent scores, but clearly a discount factor of 0.7 or 0.8 is better on average.

The main reason I believe this occurs is due to selecting players that are significantly better than other players at their positions when they are put up against other players versus their position, even if the players at another position are better overall. In other words, if there is a significant gap between two players of the same position, and a small gap between two players of another position, it may be optimal to select the player that is significantly better than his counterpart, even if the player with a smaller gap has a slight edge overall. This is due to looking ahead to the draft and understanding that there may be a plethora of players to choose with similar value at one position, so it is better to choose the player that compares well to his position right now, since all positions must be drafted eventually.

Another theory that may or may not contribute is that players at certain positions have overlapping skill sets. In other words, first basemen may be more likely to hit for power, whereas middle infielders may be more likely to hit for average. Thus, by discounting a position after selecting it, the user is getting a more diversified team categorically, and this may lead to a better overall team.

In the future, I would like to continue testing the theory that diversifying categories is an optimal way to select players. Would it be better to maximize two or three categories and ignore the rest? Or to try to keep each at about an equal level? In order to test this, I will need to add a heuristic to my model that weights each category based on my current standing in the league, and see how that standing changes based on who I may select. I can then compare outcomes where I weight categories I am already doing well in more highly versus weighting them less than categories I am lagging, and compare the results.